p–ISSN: 2723 - 6609 e-ISSN: 2745-5254
Vol. 6, No. 2 February 2025 http://jist.publikasiindonesia.id/

Shellcode *Classification* with *Machine Learning* Based on Binary Classification

Jaka Naufal Semendawai[1*], Deris Stiawan[2], Iwan Pahendra[3]
Universitas Sriwijaya, Indonesia
Email: jaka.semendawai@gmail.com, deris@unsri.ac.id,[3]iwanpahendra@unsri.ac.id

*Correspondence

Keywords: binary classification; cyber security; machine learning; supervised machine learning; hyperparameter tuning

ABSTRACT

The Internet can link one person to another using their respective devices. The internet itself has both positive and negative impacts. One example of the internet's negative impact is malware that can disrupt or even kill a device or its users; that is why cyber security is required. Many methods can be used to prevent or detect malware. One of the efforts is to use machine learning techniques. The training and testing dataset for the experiments is derived from the UNSW_NB15 dataset. K-Nearest Neighbour (KNN), Decision Tree, and Naïve Bayes classifiers are implemented to classify whether a record in the testing data is Shellcode or non-Shellcode attack. The KNN, Decision Tree, and Naïve Bayes classifiers achieve accuracy levels of 96.82%, 97.08%, and 63.43%, respectively. The results of this research are expected to provide insight into the use of machine learning in detecting or classifying malware or other types of cyber attacks.

Introduction

Awareness of the importance of *cyber security* in Indonesia is still very low. This is evidenced by data published by *the International Communication Union* (ITU), where the level of *cyber security* in Indonesia is ranked 70th. This states that Indonesia is very vulnerable to cyber attacks from *hackers* in other countries. In addition, according to data from *treat exposure rate* (TER), Indonesia has an attack vulnerability rate of *malware* by 23.54% (Ashari, 2020).

(Patterson et al., 2023) Stated that any organization must think about the essence of *cyber security*. This is due to the increasing number of attack cases, which must be able to be resisted by the knowledge of *cyber security* because it is already concerned with data privacy and infrastructure resilience issues. This can be seen from the research conducted by (Singelton et al., 2021) In 2021, ransomware attacks were used against 10 different types of companies, accounting for an average of 17.4% of the total types of attacks that occurred in these companies.

One *malware* currently trending to be used as an attack tool is *Shellcode*. Using *Shellcode*s in cyber attacks has become a trend among *hackers*. *Shellcode* can carry out illegal activities such as DoS attacks, data theft, and automatic system destruction on the destination computer. (Yang et al., 2022) *Shellcode* is code designed to perform its tasks automatically. *It* can grant an *attacker permission* to exploit the destination computer thoroughly. *Shellcode is* generally produced using *assembly*.

The basic structure of the *Shellcode* is as follows: The first is *No Operation Instructions* (NOP *Sled*). NOP *sled* is used to ensure that the execution does not fail. Then, *Bootstrap Code* is used to set up the execution environment. *Bootstrap code* usually consists of a value code *register* used by the *payload*. Next is *payload*. *The payload* used to perform the main tasks of a *hacker* depends on the code written in it. The latter is the *clean-up code*. *Clean-up code* removes traces from *hackers* after attacking the target system. This can make *Shellcode* and can provide performance from *Shellcode* in executing cleanly. Inside *Shellcode,* There are several functions referred to as *root shell*. It was the most widely used before some further developments. (Anley et al., 2007)(Niiranen, 2021).

Shellcode is developed with many tools that have their functions. Functions of the tools used to create *Shellcode* consist of tools for writing *code*, *compiling* code, *converting*, *testing*, and *debugging Shellcode*. Some of these tools can facilitate the formation or development of *Shellcode*. Some tools used to form *Shellcode* are *NASM, GDB, ObjDump, Ktrace, Strace, and Readelf*. *NASM* is a tool consisting of an *assembler*, which is called *NASM*, and a *disassembler*, which is called *NDISAM* (Foster et al., 2005)(ALHusayn, 2020)*Shellcode* can download and execute *malware* automatically when *Hackers* successfully enter the destination system. However, the detection of *Shellcode* has not been widely developed.

Research from (Akabane et al., 2019) Which develops methods to prevent the occurrence of activities *Shellcode* with the results of their research, namely *EAF Guard Driver, which is* enough to provide impressive results for prevention *Shellcode*. *The driver* can prevent the *Shellcode* well without a *false alarm rate*. Also, the test results *benchmark* from *EAF Guard Driver* increased by 0.02%.

In addition, there is research conducted by (Kanemoto et al., 2019) They use *shellcode emulation,* which is based on accuracy and performance values. They aim for a model to identify critical alerts and automatically provide information about security breaches. This study's result is the accuracy and performance obtained, which is approximately 60% *remote shellcode* Detected.

This research was conducted using *machine learning* berbasis *supervised machine learning*. Similar research was also conducted by (Moon et al., 2022) that uses *supervised machine learning* to detect *malware*. In the research, they used *feature hashing* because it can save up to 70% of the memory used and improve malware detection accuracy. Research conducted by (Rajesh Bingu, 2023) Classification based on *binary* and *multiclass classification* with the help of several *machine learning models*. The results

obtained from the study are based on binary classification, resulting in accuracy values from 99.17% to 99.65%.

In this study, researchers used several types of *machine learning*: *K-Nearest Neighbors, Decision Tree*, and *Naive Bayes*. Research conducted by (Gouda et al., 2024) The *decision tree* and *k-nearest neighbors were used* to detect *malware,* and the *data set was used, UQ-NIDS-V2*. Di dalam *Data Set* There was an attack *Shellcode*. The study results show that using the *decision tree* cannot detect attacks on *Shellcode*. This can be seen from the precision, sensitivity, and *F1 Score,* which is 0%. However, for accuracy values in detecting all types *of malware,* which is 98.78%. Then, for the model *K-nearest neighbors*, the detection of *Shellcode is* also not good. This can also be seen from the precision, sensitivity, and *F1 Score,* which is 0%. Moreover, accuracy values in type detection *malware* are at 98.16%. In addition, research conducted by (Samantaray et al., 2024) They use different types of *machine learning,* such as SVM, KNN, LR, DT, NB, and RF. Then, the model uses an algorithm called *MaxAbsScaler*. The results of this study are that the accuracy value obtained in carrying out classification based on *multiclass classification* increased from 60% to 94% with the help of engineering *MaxAbsScaler*.

This research uses machine learning models to introduce an optimized classification approach for detecting Shellcode attacks. Unlike previous studies focusing solely on feature extraction or traditional anomaly detection methods, this study integrates hyperparameter tuning and performance evaluation to identify the most effective classification model. The use of multiple classification techniques in a comparative framework enhances the novelty of this study. Given the increasing frequency of cyberattacks and Indonesia's vulnerability to cybersecurity threats, developing efficient and reliable detection systems is crucial. The findings of this study can contribute to strengthening national cybersecurity strategies and protecting digital assets from malicious intrusions.

In this study, we used machine learning, namely K-nearest neighbors, decision trees, and naïve Bayes. This is because the three models can be used to perform binary-based classifications. Then, from the test results using the three models, we will take data such as accuracy, F1-Score, precision, and recall, which we will then analyze. In the KNN model, we use various kinds of tests, namely data scaling and hyperparameter tuning, to evaluate the data that has been improved. In addition, data scaling or hyperparameter tuning is only used on data, and data scaling or hyperparameter tuning is not used. We also do this for the decision tree model. Moreover, we also did the naïve Bayes model but varied the comparison of training and testing data.

This study aims to evaluate the performance of KNN, Decision Tree, and Naïve Bayes classifiers in detecting Shellcode attacks, analyze the impact of hyperparameter tuning on classification accuracy, and compare the efficiency of different machine learning models in identifying cyber threats.

The expected benefits of this study include providing a reference for future research on machine learning applications in cybersecurity, offering insights into effective

Shellcode detection techniques for cybersecurity professionals, and supporting the development of policies and regulations to enhance cybersecurity resilience.

Research Methods

This study uses a qualitative research type with a descriptive approach to analyze the impact of regulations on the opening of flight routes from and to Kertajati Airport. This research examines Shellcode attacks using binary classification to see the performance of *classifier KNN, Decision Tree, and Naïve Bayes* in detecting attacks on *Shellcode*. The data used in this study was taken from the UNSW_NB15 dataset, where many studies have been conducted on the dataset with the type of attack *Shellcode*. The rationale behind using the UNSW-NB15 dataset is that it has up-to-date, logical, and feature data from each attack that can provide access to analyze each attack technique. (Moustafa et al., 2018). The steps that will be taken in this study are as follows.

1. Create a new dataset by filtering Shellcode and non-Shellcode attack types from the UNSW_NB15 dataset.
2. Label the data using labels 0 and 1, where label zero is for *non-Shellcode* attacks and label 1 is for *Shellcode attack types*.
3. Split the data for the training dataset and the test dataset.
4. Test the data using *the KNN, Decision Tree,* and *Naïve Bayes models*.
5. Compare the performance of the KNN, Decision Tree, *and* Naïve Bayes *classifiers* regarding non-shellcode attack detection accuracy vs. *Shellcode attacks.*
6. Analyze the results that have been obtained from the experiment.

Data Preprocessing

Before we test the data, we first preprocess the data. After we get the CSV file from the dataset, UNSW_NB15, we select the features using the Exploratory Data Analysis (EDA) method. Then, once we get the best features of the EDA method, we create a new CSV file containing the data with the selected features of the EDA method. After that, we scale the data for multiple tests using the StandardScaler. The results of the previous steps produce a dataset that is ready to be tested. The graph of the pre-processing data will be shown in Figure 1 below:

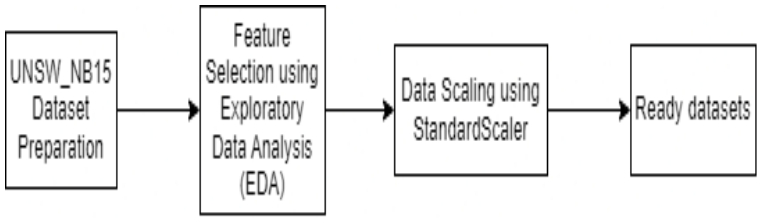

Figure 1. Data Preprocessing Process

Research Flow

In the first step of the research, the author conducted a literature study to find some previous research that supports this research. Then, several publicly available datasets were investigated to determine the dataset that best suited the experiment's needs. After

careful investigation, the UNSW_NB15 dataset was selected. Some important attributes of the traffic records in the dataset are then considered features. We use the *Exploratory Data Analysis* method to select the best features. Multiple CSV files from a dataset that have been processed are combined into a single CSV file. For some tests, we scale the data using the StandardScaler method. In addition, we also use hyperparameter tuning for each machine learning to get good results from the tests. Data separation for training and testing purposes uses machine learning libraries available in the Python programming language. Furthermore, experiments using *KNN, Decision Tree,* and *Naïve Bayes classifiers* were carried out, and the results were further analyzed. The workflow of the proposed method is illustrated in Figure 2.

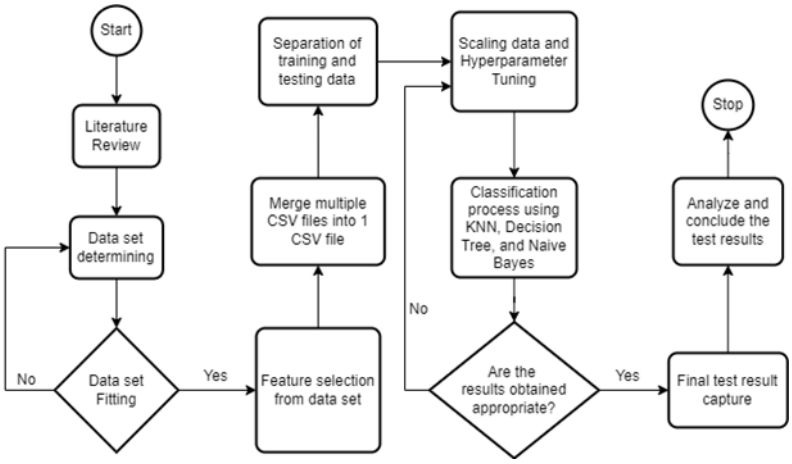

Figure 2. Research Flow

Feature Selection from UNSW_NB15 Dataset

The UNSW_NB15 dataset has 49 features. These features have their functions. This feature was obtained from the results of previous research, where researchers previously took data using the Tcpdump application to see the traffic that occurred when the experiment was run. In this study, the author only took a few features suitable for the experiment to get optimal results and meet the study's objectives. We use Exploratory Data Analysis (EDA) to get the best features for our research. Table 1 describes the selected features.

Table 1. Description of Dataset UNSW_NB15 Features

Number	Feature	Type	Description
1.	stop	Float	Record total duration
2.	bytes	Integer	Source to destination transaction bytes
3.	bytes	Integer	Destination to source transaction bytes
4.	Sload	Float	Source bits per second
5.	means	Integer	Mean of the flow packet size transmitted by the source
6.	means	Integer	Mean of the flow packet size transmitted by the destination
7.	Estimates	Timestamp	Record start time

Number	Feature	Type	Description
8.	Sintpkt	Float	TCP connection setup time is between the SYN and the SYN_ACK packets.
9.	label	Binary	0 for non-shellcode and 1 for shellcode records

KNN *Classifier*

The KNN method classifies based on learning by analogy. The KNN algorithm can find patterns for the k-nearest value. The value of k is the neighbor k of the value of an unknown sample. The degree of "proximity" is described in Eudian distance terms. Eudian distance is the distance between and, which will be written in (3) $X = (x_1, x_2, \dots, x_n) Y = (y_1, y_2, \dots, y_n)$(Han & Kamber, 1998).

$$d(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad (1)$$

Using *K-Nearest neighbors* for data classification begins with merging raw *files* from the *dataset*. Then, from the raw data, features suitable for the classification process will be selected using the *exploratory data analysis* method. Then, the researcher used *an oversampling* technique to obtain a balanced dataset. Furthermore, *training* and *testing data will be separated*. In this research, the author uses a ratio variation, namely 90:10, 80:20, 70:30, 60:40, and 50:50, for training and *testing* data. For classification using KNN, researchers use *the hyperparameter tuning* method to get optimal performance. Then, the data visualization from the KNN model will be displayed as a graph of accuracy values and the *F1-Score* of the entire test. After obtaining the classification results in the form of *an F1 score* and determining the level of accuracy, the author will analyze and conclude from the results.

Decision Tree Classifier

Decision Tree Classifier is a process of classifying data by changing the form of data (tables) into tree models, changing tree models into rules, and simplifying rules (trimming). A statistical property called information gain is used to determine the best attributes. Information gain measures how reliable an attribute is in separating training samples according to their target classification. To determine the exact information gain, we start by selecting a measure called entropy in information theory, which characterizes the purity/impurity of a random sample set. (Sarimuddin et al., 2020).

The use of *decision trees* for data classification begins with the stage of merging raw *files* from the datasets used. Then, the raw data will be selected using the Exploratory Data Analysis method to select features suitable for the classification process. Then, the researcher used *an oversampling* technique to obtain a balanced dataset. Furthermore, *training* and *testing data will be separated*. In this research, the author uses a variation of the ratio, namely 90:10, 80:20, 70:30, 60:40, and 50:50, for training and *testing* data. The researcher uses *the hyperparameter tuning* method in the decision tree model to get more optimal performance. Then, the authors used *the graphviz* model to display the decision tree from the *dataset*. After the data visualization is carried out as a decision tree, the data

visualization from the DT model will be displayed as an accuracy value graph and *F1-Score* of the entire test. After obtaining the classification results in the form of *an F1 score* and determining the level of accuracy, the author will analyze and conclude from the results.

Naïve Bayes *Classifier*

Two stages are needed if we use the *classifier Naïve Bayes* to classify data. First, make *a bag of words*, then continue by conducting training. This is done to get a classification model in the form of probability. Furthermore, to maintain the test results so they are not damaged, *Laplace Smoothing is necessary;* up to 0 chance of *training* can be avoided. *Laplace smoothing* Adding the number 1 is divided by the sum of all features added to all features so that nothing is worth 0. The Naïve Bayes equation used to determine the class will be written in (4) (Fitriyyah et al., 2019).

$$V_{MAP} = argmax_{v_j} evP(v_j) \prod_{i=1}^{n} P(a_j|v_j) \qquad (2)$$

Naïve Bayes for data classification begins with combining *raw files* from the datasets used. Then, the raw data will be selected using the Exploratory Data Analysis method to select features suitable for the classification process. Then, the researcher used *an oversampling* technique to obtain a balanced dataset. Furthermore, *training* and *testing data will be separated*. In this research, the author uses a variation of the ratio, namely 90:10, 80:20, 70:30, 60:40, and 50:50, for training and *testing* data. In the classification using *Naive Bayes*, the researcher used *Gaussian Naive Bayes*. After the classification process is carried out using *naïve Bayes*, the results of the *F1-Score classification*, and the level of accuracy, the author will analyze and draw conclusions from the results. The data visualization of these test results will be displayed in a graph of each test's accuracy value and *F1-Score*.

Confusion Matrix

The Confusion matrix contains actual values and classification predictions used to evaluate classifications and predict correct or false objects. (Abdillah, 2015). The *confusion matrix* will be shown in Table 2.

Table 2. Confusion Matrix

Classification	Predicted Class	
	Shellcode	Non-shellcode
Shellcode	True Positive (TP)	False Negative (FN)
Non-shellcode	False Positive (FP)	True Negative (TN)

Based on the confusion matrix table above, the calculation of the accuracy value can be done using the formula (3) as follows:

$$Accuracy = \left(\frac{TP+TN}{TP+FP+TN+FN}\right) * 100\% \qquad (3).$$

Results and Discussion

This research was conducted on a computer with the following specifications: 8 GB RAM, Intel Core i5-8520U processor, and Windows 10 operating system. The classifier is implemented on the *Google Colaboratory platform*. The results of the research will be presented in the following section.

Research Results

This study uses three binary classifications: KNN, Decision Tree, and Naïve Bayes (NB).

Classification with K-Nearest Neighbors

The results of the Classification by KNN will be displayed through the following image:

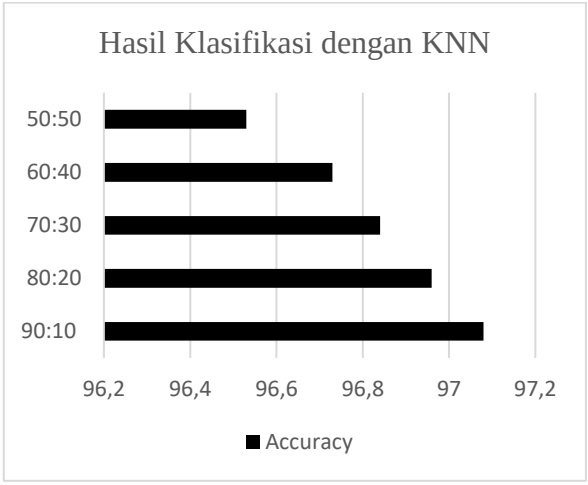

Figure 2. Accuracy Level of Classification Results with KNN.

Based on the figure above, the *K-Nearest Neighbors* model performs well in classifying this data set. This can be seen from several test results using the KNN method. From the test results that used a training and testing data ratio of 90:10. This test uses the StandardScaler and Hyperparameter Tuning methods, which results in an accuracy value of 97.08% and an F1-Score of 97.07%. This value is the best produced by the test using the ¬K-Nearest Neighbors method. Then, the parameters that provide the best performance are obtained using hyperparameter tuning, where these parameters are metric = Euclidean, n_neighbors = 4, and weight = distance. Thus, the maximum performance of KNN is obtained when using the value K = 4.

Classification with Decision Tree

Then, the *decision tree* model is a method that produces the best performance in classifying *shellcode* attacks. The test that produces the best performance is when performing *hyperparameter tuning*. This test compares *training* and *testing* data of 90:10,

with an accuracy value of 97.22%. Then, the *F1-Score* value generated by this test is 97.22% for the classification of *shellcode* attacks. This proves that this model can classify *shellcode* attacks. Then, by performing *hyperparameter tuning* on the *decision tree* model, the best parameters produced are *criterion*: *entropy* and *random state* = 42 with *an F1-Score* of 97.22%. The following image will display the classification results with *the decision tree*.

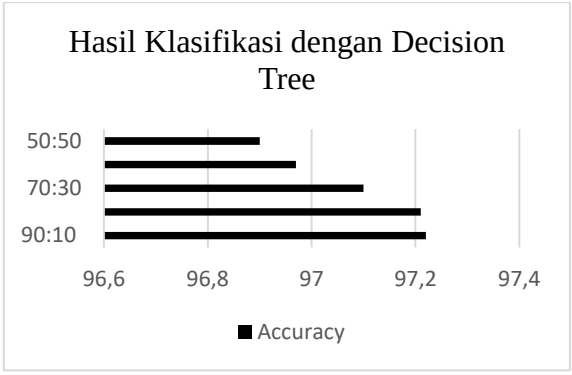

Figure 3. Accuracy Level of Classification Results with *Decision Tree*

Classification with Naïve Bayes

The following figure will show the results of the Classification with Naïve Bayes.

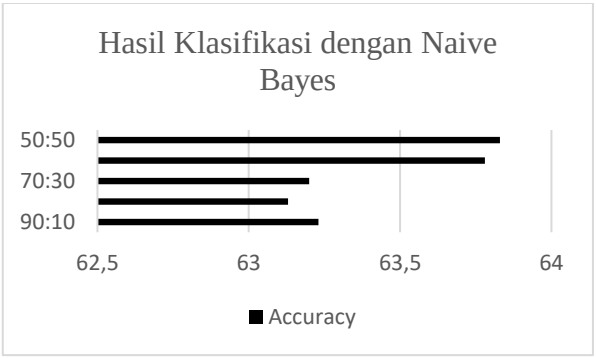

Figure 4. Accuracy of Classification Results with Naïve Bayes.

Then the last model used in this study is Naïve Bayes. This model produces performance that is not as good as the KNN and decision tree models. This can be seen from the tests that perform best, namely when comparing training and testing data of 50:50. This test also carried out the Hyperparameter Tuning stage, which resulted in an accuracy value of 63.83%. In addition, the F1-Score resulting from this test is 51.93%. This proves that this model can classify shellcode attacks and non-shellcode attacks. However, this model is unreliable in classifying data from shellcode attacks. Then, by performing hyperparameter tuning on Gaussian Naïve Bayes, the best performance of this model is achieved, where the parameter is var_smoothing = 101 with an F1-Score of 0.613.

Discussion

Table 3. Accuracy and F1-Score value from all *Machine Learning*

Method	Variation	Accuracy (%)	F1-Score (%)
K-Nearest Neighbors	90:10	97	97
	80:20	97	97
	70:30	97	97
	60:40	96	97
	50:50	96	98
Decision Tree	**90:10**	**98**	**97**
	80:20	98	97
	70:30	97	97
	60:40	97	97
	50:50	97	97
Naïve Bayes	90:10	77	37
	80:20	78	37
	70:30	77	37
	60:40	77	39
	50:50	77	39

Source: Data processed

Table 3 above shows that the K-Nearest Neighbor and Decision Tree classification results have the best performance. However, the overall performance of *the KNN* classifier in the *Shellcode attack classification* is not good enough. This is due to the characteristics of the KNN classifier, which is resistant *(robust)* to extreme data changes *(outlier)* where the data used for testing has significant changes. However, this classification is not optimal for sizable data even though it resists significant changes. This is because if the data change is too substantial, then this change will be a weakness for this *classifier*. So, there is a limit to the data change in the *KNN classifier*. As for the *Decision Tree classifier*, it also performs relatively well in *Shellcode* classification. This good achievement is also influenced by the advantages of *the Decision Tree* model, which is not sensitive to significant changes *in data (outliers).* In addition, *the Decision Tree* model also has a pretty good accuracy value in classifying and predicting data. *The Naïve Bayes CLassifier* also performed well in classifying and predicting during test experiments. However, this *classifier* is not good enough in classifying *Shellcode attacks*; it is not even recommended. This is due to the characteristics of *classifiers* that have limited performance for complex data. In addition, this model is also quite sensitive to the features used. Overall, the visualization of the classification data used in the test shows

that all three *machine learning-based classifiers* perform relatively well in detecting positive data compared to harmful data.

Conclusion

This study analyzes the *classification of Shellcode* with *machine learning* based on binary classification, namely KNN, Decision Tree, and Naïve Bayes. Overall, all three *classifiers* performed well in classifying *Shellcode* attacks. The *Decision Tree* classifier achieved the best accuracy level of 97.21%. However, the accuracy of the KNN and Naive Bayes methods also showed results that did not disappoint. This indicates that binary classification-based classification can classify *Shellcode* attacks taken from the UNSW_NB15 dataset. The results of this study are expected to provide insights into the use of *machine learning* in detecting or classifying *malware* or other types of cyberattacks. Further research can explore *preprocessing* and/or classification methods with multiclass classification to detect and classify types of cyber attacks, especially *Shellcode Attacks*. In addition, the types of attacks can be more varied and not limited to the datasets used in this study.

Bibliography

Abdillah, S. (2015). Penerapan Algoritma Decision Tree C4.5 Untuk Diagnosa Penyakit Stroke Dengan Klasifikasi Data Mining Pada Rumah Sakit Santra Maria Pemalang. *Jurnal Teknik Informatika*, 1–12.

Akabane, S., Miwa, T., & Okamoto, T. (2019). An EAF guard driver to prevent shellcode from removing guard pages. *Procedia Computer Science*, *159*, 2432–2439. https://doi.org/10.1016/j.procs.2019.09.418

ALHusayn, S. M. S. (2020). The Buffer Overflow Attack and How to Solve Buffer Overflow in Recent Research. *AJRSP Journal*, *2*(19), 1–13.

Anley, C., Heasman, J., Linder, F., & Richarte, G. (2007). *The Shellcoder's Handbook: Discovering and Exploiting Security Holes, 2nd Edition*.

Ashari, M. (2020). *Keamanan Informasi: Sudah Saatnya Kita Peduli*. DJKN Kemenkeu.

Fitriyyah, S. N. J., Safriadi, N., & Pratama, E. E. (2019). Analisis Sentimen Calon Presiden Indonesia 2019 dari Media Sosial Twitter Menggunakan Metode Naive Bayes. *Jurnal Edukasi Dan Penelitian Informatika (JEPIN)*, *5*(3), 279. https://doi.org/10.26418/jp.v5i3.34368

Foster, J. C., Osipov, V., Bhalla, N., Heinen, N., & Aitel, D. (2005). Buffer overflow attacks: Detect, exploit, prevent. In *Buffer Overflow Attacks: Detect, Exploit, Prevent*. https://doi.org/10.1016/B978-1-932266-67-2.X5031-2

Gouda, H. A., Ahmed, M. A., & Roushdy, M. I. (2024). Optimizing anomaly-based attack detection using classification machine learning. *Neural Computing and Applications*, *36*(6), 3239–3257. https://doi.org/10.1007/s00521-023-09309-y

Han, J., & Kamber, M. (1998). Data Mining: Concepts and Techniques. In *Morgan Kaufmann Publisher*. https://doi.org/10.3726/978-3-653-01927-8/2

Kanemoto, Y., Aoki, K., Iwamura, M., Miyoshi, J., Kotani, D., Takakura, H., & Okabe, Y. (2019). Detecting successful attacks from IDS alerts based on emulation of remote shellcodes. *Proceedings - International Computer Software and Applications Conference*, *2*, 471–476. https://doi.org/10.1109/COMPSAC.2019.10251

Moon, D., Lee, J. K., & Yoon, M. K. (2022). Compact feature hashing for machine learning-based malware detection. *ICT Express*, *8*(1), 124–129. https://doi.org/10.1016/j.icte.2021.08.005

Moustafa, N., Adi, E., Turnbull, B., & Hu, J. (2018). A New Threat Intelligence Scheme for Safeguarding Industry 4.0 Systems. *IEEE Access*, *6*, 32910–32924. https://doi.org/10.1109/ACCESS.2018.2844794

Niiranen, A. (2021). *Machine learning-based ISA detection for short shellcodes*.

Patterson, C. M., Nurse, J. R. C., & Franqueira, V. N. L. (2023). Learning from cyber security incidents: A systematic review and future research agenda. *Computers and Security*, *132*. https://doi.org/10.1016/j.cose.2023.103309

Rajesh Bingu, E. al. (2023). Performance Comparison Analysis of Classification Methodologies for Effective Detection of Intrusions. *International Journal on Recent and Innovation Trends in Computing and Communication*, *11*(9), 2860–2879. https://doi.org/10.17762/ijritcc.v11i9.9375

Samantaray, M., Barik, R. C., & Biswal, A. K. (2024). A comparative assessment of machine learning algorithms in the IoT-based network intrusion detection systems. *Decision Analytics Journal*, *11*(May), 100478. https://doi.org/10.1016/j.dajour.2024.100478

Sarimuddin, S., Sari, J. Y., Mail, M., Masalu, M. A., Aristika, R. S., & Nurfagra, N. (2020). Klasifikasi Data Aging Tunggakan Nasabah Menggunakan Metode Decision Tree Pada ULaMM Unit Kolaka. *INFORMAL: Informatics Journal*, *5*(1), 26. https://doi.org/10.19184/isj.v5i1.16964

Singelton, C., Wikoff, A., & McMillen, D. (2021). IBM: 2021 X-Force Threat Intelligence Index. *Network Security*, 36. https://doi.org/10.1016/s1353-4858(21)00026-x

Yang, G., Chen, X., Zhou, Y., & Yu, C. (2022). DualSC: Automatic Generation and Summarization of Shellcode via Transformer and Dual Learning. *Proceedings - 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022*, 361–372. https://doi.org/10.1109/SANER53432.2022.00052